Rapport

Projet OpenGL

ROUCKETLEAGUE

IMAC 1

Matthieu BESSOL & Antoine DEMIERE & Lisa COUAVOUX

Sommaire

1. Présentation du projet 3
2. Difficultés rencontrées
3. Architecture du programme

Éléments demandés et codés qui fonctionnent :

* checkpoints présents sous forme de cercle
* l’aéroglisseur a une physique avec rotation, accélération et collision
* les deux aéroglisseurs sont contrôlables aux claviers (fllèches et zqsd)
* collision avec le checkpoint visible grâce à sa disparition et à un changement des propriétés du véhicule (boost ou freeze).
* le terrain est généré à partir d’un fichier texte où l’on définit les murs, la position des checkpoints et sa dimension.
* le jeu devant permettre de voir toute la carte, nous ne donnons pas la possibilité au joueur de zoomer sur son véhicule. Cependant, lorsqu’il y a un but, nous zoomons sur la balle et faisons en sorte que la caméra ne montre que l’intérieur du terrain. De même, nous ne pouvons pas afficher les checkpoints non parcourus sans le zoom sur le joueur. Nous avons quand même placé une mini-carte pour voir l’évolution du jeu.

Éléments demandés et codés qui ne fonctionnent pas :

Éléments demandés mais pas codés :

Éléments non demandés et codés qui fonctionnent :

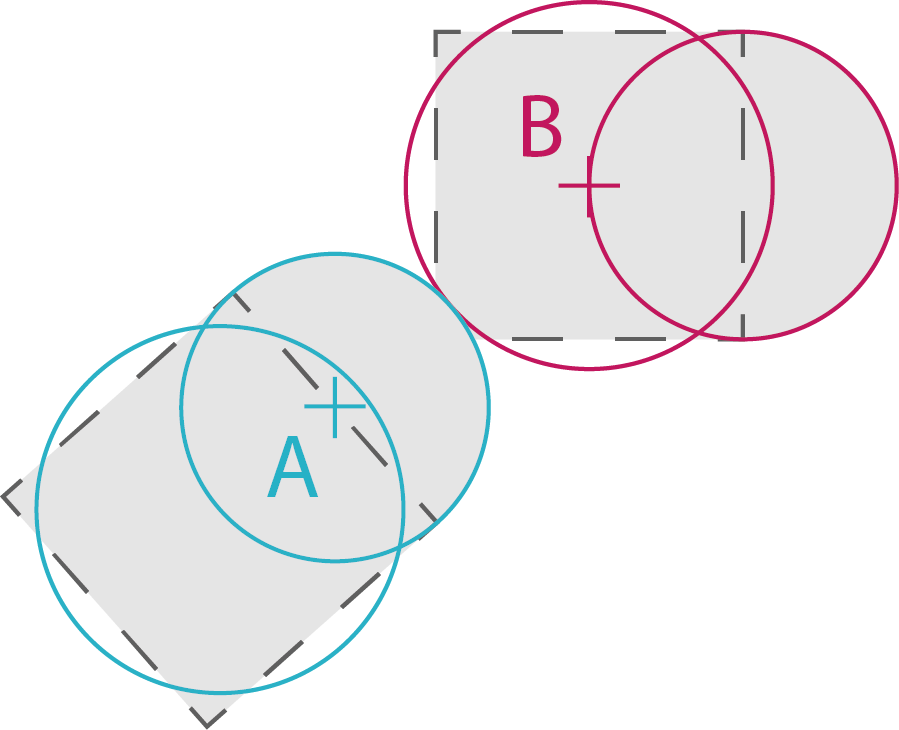
* Un menu a été créé, il permet de créer une nouvelle partie ou de quitter le jeu.
* Du son a été rajouté avec SDL\_mixer.
* Une identité visuelle a été créée.
* Collision entre deux véhicules.
* Rajout d’une balle avec sa physique permettant de marquer des buts.
* Système de score.
* Timer pour les checkpoints.
* Ralenti des véhicules lorsqu’un but est marqué

Présentation du projet

Nous avons décidé de modifier le jeu initial du projet. Ce ne sera pas une course contre la montre mais un jeu de football entre deux Hovercraft, à la manière de Rocket League, le but étant de marquer plus de but que son adversaire dans un temps imparti. C’est donc un jeu multi-joueurs jouable à 2 sur un clavier (Z-Q-D pour le joueur 1 et les flèches directionnelles pour le joueur 2).

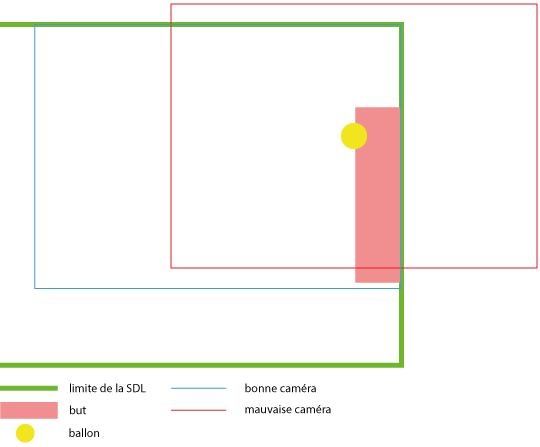
Il a alors été nécessaire de gérer les collisions entre les véhicules, le ballon, le terrain, le but ainsi que les checkpoints. Ces derniers ont été gardés du sujet initial et correspondent à des bonus (accélération de son propre véhicule) ou des malus (« freeze » du véhicule adverse).

**Difficultés rencontrées**

Une des difficultés principales rencontrée dans le développement du jeu a été le système de collision entre les véhicules. Nos véhicules sont composés d’un carré et d’un cercle à l’avant. Nous n’en avions jamais fait auparavant et ne voyions pas comment faire pour gérer les collisions entre un cercle et un carré par exemple. Nous avons fait différentes tentatives mais avons vite vu qu’elles étaient trop compliquées. Nous avons remarqué cependant que les collisions entre deux cercles étaient beaucoup plus simples à coder et moins gourmandes en calcul. Nous avons donc décidé de créer un faux cercle autour du carré du véhicule qui nous servirait à effectuer les collisions.

Ainsi, si la distance entre le point A et le point B (les centres des deux cercles qui vont entrer en collision) est inférieur ou égale à la somme des rayons des deux cercles, alors il y a un contact. Cette méthode permet un nombre minime de calcul. Les collisions sont cependant moins précises.

Une autre difficulté a été de faire en sorte que la caméra ne dépasse pas le terrain lorsque l’on marque un but.

Pour calculer la bonne distance il a fallut alors faire attention au zoom qui augmentait au fur et à mesure sur le ballon ainsi qu’à la position du ballon et des extrémités de la caméra. Après différents tests, nous avons finalement trouver la bonne formule qui permet de délimiter la position de la caméra au sein du jeu. 

Architecture du programme

**Architecture Principale :**

**Main**

**Menu**

Loop Menu

Play Game (Game.c)

**Play Level** (Level.c)

Lancement du chrono (5 minutes pour 1 partie)

Loop Level

Mis à jour des véhicules (position ect)

Vérification des collisions (Collision.c)

**Architecture des Véhicules**: (Vehicule.c)

|  |  |
| --- | --- |
| typedef struct {  float hauteur;  float largeur;  Cercle\* cercle;  Cercle\* facticeCercle;  float angle;  Point2D position;  Vector2D direction;  Vector2D vitesse;  Vector2D acceleration;  int avance, tourne;  Player player;  Bonus bonus;  int timerBonus;  }Vehicule; | **Automate :**  TOURNE  Renvoie 0 si aucune touche n’est pressée  Renvoie -1 si touche pour aller à gauche pressée  Renvoie 1 si touche pour aller à droite pressée  AVANCE  Renvoie 0 si aucune touche n’est pressée  Renvoie 1 si touche pour avancer pressée  **Physique du véhicule**  Direction : vecteur unitaire initialisé selon l’orientation du véhicule  Rotation : modifie la direction du vecteur « direction » du véhicule  Accélération : Vecteur colinéaire à la direction, de norme plus grande  (Vérifie la position de l’automate (Avance = 1 ou Avance = 0)  Vitesse : Vecteur additionnel de lui-même avec le vecteur accélération.  On lui enlève ensuite une friction pour se rapprocher du modèle  physique.  Position : point qui permet de faire avancer de manière visuelle le véhicule  suivant les paramètres de rotation, accélération et vitesse. |

**Architecture du Ballon:**

|  |  |
| --- | --- |
| typedef struct {  GLuint texture;  Vector2D direction;  Vector2D vitesse;  Vector2D acceleration;  Cercle\* cercle;  }Ballon; | **Physique du ballon**  Basé sur le même principe que celle du Véhicule, à l’exception près que la direction du ballon se fait en fonction des collisions avec le véhicule ou le terrain. Dès qu’une collision a lieu, un vecteur unitaire est créé entre les centres des deux cercles intersectés. (en noir sur le schéma)    Une accélération colinéaire à la direction est créée dès que la collision a lieu. Elle influe ensuite la vitesse du ballon et enfin sa position qui sont mis à jour grâce à la boucle du programme. |